2022-06-23

Inlämningsuppgift i kursen Grundläggande Programmering i Python, sommar-2022

Syfte och mål

Syftet med inlämningsuppgiften är att du ska visa, inte minst för dig själv, att du kan skapa ett Python-program utifrån en definierad problembeskrivning där flera av de färdigheter som du fått i kursen används. Målet med uppgiften är att du, efter slutförd uppgift, har fått ytterligare ökad förståelse för programmeringens grunder och att du fått insikt om hur man bryter ned en programmeringsuppgift i mindre delar.

Förutsättningar

All administrativ information om datum för inlämning, hur kamratgranskningen går till, betyg & rättning, osv finner du på Uppgifts-sidan i Canvas.

Uppgiften utförs i form av 6 st. deluppgifter, där den sista uppgiften blir att sätta ihop programmen från de olika uppgifterna och skapa ett sammanhållet program via ett enkelt menysystem.

Varje deluppgift bedöms var för sig med ”OK” eller ”ej OK”. För ”OK” krävs att den är *acceptabelt* korrekt utförd, och för att inlämningsuppgiften i helhet ska bli godkänd vid lärarinlämningen i slutet (se flödesdiagrammet på Uppgifts-sidan i Canvas) måste alla deluppgifter vara bedömda ”OK”.

En grundregel i lösandet av denna inlämningsuppgift är att du inte får använda färdiga funktioner i moduler som går att importera, för att ev. försöka förenkla några beräkningar. Men med undantag för följande moduler som är helt nödvändiga och/eller ni särskilt ska träna på att använda:

- modulen *CSV* (se kapitel 10 i Canvas) för att hantera CSV-filer
- modulen *matplotlib.pyplot* (se kapitel 11 i Canvas) för plottning
- modulen *datetime* (se kapitel 9 i Canvas) för hanterandet av tider/klockslag

Vi ställer inga krav på att ”inmatningsfelkontroller” måste finnas, t.ex. på deluppgift 6 där användaren skulle kunna mata in ogiltiga menyval.

Några allmänna riktlinjer som skall följas:

- koden ska vara genomtänkt, lättläst och välkommenterad
    - skriv varför kodrader finns, och inte vad de tekniskt gör. Se i kursmaterialet för diskussion och exempel på detta (kapitel 2 i Canvas *Introduktion och ett första programexempel*)
- variabler ska ha meningsfulla namn
- du får inte lämna in kod som genererar felmeddelande eller varningar
- du får inte lämna kvar större sektioner med gammal ej aktuell bortkommenterad kod (gör koden svårare överskåda)

Programkoden skriver du i Jupyter Notebook - använd den tillhandahållna förbereda svarsfilen *”Svarsfil_inluppg_A237TG.ipynb”*. Denna Jupyter Notebook-fil med dina lösningar blir det enda du laddar upp vid inlämning i Canvas (dvs tillhörande .csv-filer behövs inte skickas in). Ändra inte namn på svarsfilen vid inlämning. Vill du kommentera dina lösningar på deluppgifter på ett allmänt sätt (ex. *”jag tänkte så här … ”, ”jag valde att lösa problemet på detta sättet pga …”* osv), så använd de förskapade *Markdown*-cellerna i svarsfilen till detta.

Uppgiftsbeskrivning

I denna inlämningsuppgift kommer du att analysera och vidareförädla data från kameraövervakningar av hastigheter i Västra Götalands län. Datan kommer från Trafikverket och innehåller ett begränsat urval av hastigheter som dessa kameror registrerat på respektive mätplats den 11 september 2021 mellan kl. 07:00 – 18:00. All nödvändig information för att lösa nedanstående uppgifter finns i csv-filerna *kameraData.csv* och *platsData.csv*. Den ursprungliga *kameraData.csv* innehöll över 700 000 poster men ni får en nedbantad, men ändå en representativ version som ”bara” innehåller ca 1 300 st. att arbeta med i stället. Det gör att ni lättare kan ”manuellt” överskåda datan för att se om era beräknande program som ni ska skriva fungerar som de ska, genom att öppna csv-filen i ett godtyckligt kalkylprogram (t.ex. *Microsoft Excel* eller *OpenOffice Calc*).

Nedan följer en beskrivning av innehållet i respektive csv-fil:

kameraData.csv *(skärmbild på de första raderna)*

	A	B	C	D	E
1	MätplatsID	Gällande Hastighet	Hastighet	Datum	Tid
2	14075010	40	55	2021-09-11	11:15:31
3	14075010	40	54	2021-09-11	08:09:17
4	14075010	40	53	2021-09-11	13:02:41
5	14075010	40	49	2021-09-11	13:02:55
6	14075010	40	47	2021-09-11	13:14:45
7	14075010	40	45	2021-09-11	07:57:27

Varje rad i filen *kameraData.csv* innehåller en kameraregistrering med följande information:

MätplatsID:	Mätplatsens identifikationsnummer
Gällande hastighet:	Högsta tillåtna hastighet (km/h)
Hastighet:	Av kameran registrerad hastighet (km/h)
Datum:	Datum för registrering
Tid:	Tidpunkt för registrering

platsData.csv *(skärmbild på de första raderna)*

	A	B	C	D
1	MätplatsID	Namn	Vägnummer	Kommun
2	14002010	Bhpl Gestadvägen	E45	Vänersborg
3	14002020	Bhpl Gestadvägen	E45	Vänersborg
4	14002030	Bhpl Flybo	E45	Vänersborg
5	14002040	Bhpl Flybo	E45	Vänersborg
6	14002050	Frändefors	E45	Vänersborg
7	14002060	Frändefors	E45	Vänersborg

Varje rad i filen *platsData.csv* innehåller följande information:

MätplatsID:	Mätplatsens identifikationsnummer
Namn:	Namn på platsen där kameran är installerad
Vägnummer:	Vägnumret för den väg där kameran är installerad
Kommun:	Kommunen där kameran är installerad

Uppgifter

Nedan följer uppgifterna. Observera att i Jupyter Notebook så kan skrivna funktioner från en tidigare kodcell (i detta fallet en deluppgift) kallas i en senare kodcell (så länge den tidigare kodcellen har blivit körd minst en gång). Det innebär att du inte behöver (och inte ska) kopiera & klistra in tidigare skrivna funktioner som ska återanvändas i senare deluppgifter, för att hålla nere mängden kod och för överskådlighetens skull.

När du är färdig bör du lägga lite tid på att gå igenom dina program och förvissa dig om att dessa ger korrekta resultat, samt dubbelkolla att koden inte bryter mot någon satt regel eller riktlinje som fanns under *Förutsättningar.*

Innan inlämning bör du även i Jupyter Notebook rensa alla variabler i minnet och testköra programmen en sista gång för säkerhets skull. Detta för att Jupyter Notebook kan ”komma ihåg” värden från gamla variabler från kod som är borttagen, vilket i vissa fall kan göra att ett program som fungerar vid provkörningen före rensning inte fungerar som tänkt nästa gång någon provkör det. Du rensar variablerna genom att välja alternativet ”Restart & Clear Output” under menyalternativet ”Kernel” (alternativt att starta om datorn och Jupyter Notebook, vilket garanterat ger samma effekt).

1. Skapa en egendefinerad funktion *read_file(file_name)* som öppnar en csv-fil med angivna namnet till parametern *file_name* och läser in dess innehåll och returnerar resultatet i en tvådimensionell lista. Tanken är att när du sen kallar på denna funktion två gånger med argumenten *kameraData.csv* respektive *platsData.csv* så kommer datan finns inläst & redo för att kunna utföra efterkommande deluppgifter. Döp dina listor till *kameradata* respektive *platsdata*. När du använder funktionen *open()* för att påbörja inläsningen av filerna, skicka med *encoding = ’UTF-8’* som argument för att säkerställa att inläsningen lyckas, och att teckentolkningen på bl.a. åäö blir korrekt *(om inte så varierar det lite hur det beter sig, beroende på operativsystem).*

Avsluta uppgiften med ett huvudprogram som anropar funktionen och sedan skriver ut de *tre* första raderna i var och en av de två listorna, för att verifiera att det funkar som det skall. Utskriften ska se ut som följande:

Kameradata-filen:

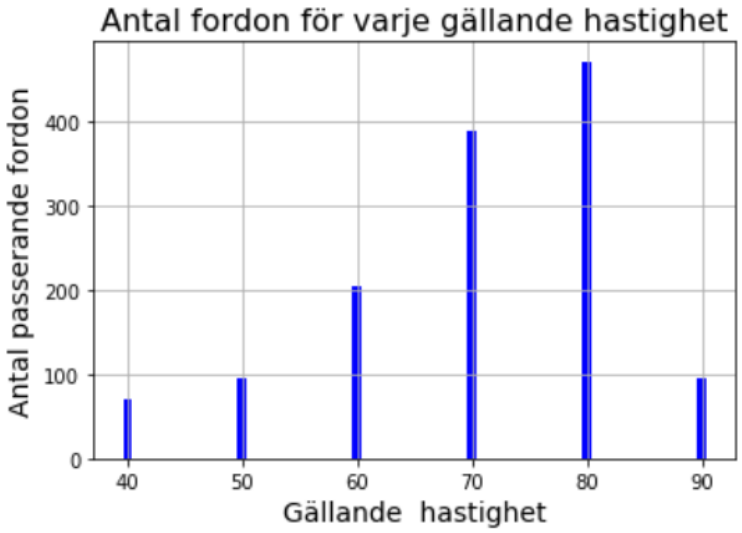
```
[['MätplatsID', 'Gällande Hastighet', 'Hastighet', 'Datum', 'Tid'],
['14075010', '40', '55', '2021-09-11', '11:15:31'], ['14075010', '40',
 '54', '2021-09-11', '08:09:17']]
```

Platsdata-filen:

```
[['MätplatsID', 'Namn', 'Vägnummer', 'Kommun'], ['14002010', 'Bhpl
  Gestadvägen', 'E45', 'Vänersborg'], ['14002020', 'Bhpl Gestadvägen',
'E45', 'Vänersborg']]
```

2. Skapa en egendefinerad funktion *antal_bilar(kamera_data)* som beräknar totalt antal fordon som passerat vid var och en av de olika gällande hastigheterna. Funktionen ska skriva ut resultatet i textform, samt även rita ett stapeldiagram över resultatet. Avsluta uppgiften med ett huvudprogram som visar att din funktion fungerar korrekt och ger nedanstående utskrift och diagram.

```
Det finns 69 mätningar där gällande hastighet är 40 km/h
Det finns 96 mätningar där gällande hastighet är 50 km/h
Det finns 204 mätningar där gällande hastighet är 60 km/h
Det finns 389 mätningar där gällande hastighet är 70 km/h
Det finns 471 mätningar där gällande hastighet är 80 km/h
Det finns 94 mätningar där gällande hastighet är 90 km/h
------------------------------------------------------------
Totalt passerade 1323 bilar.
```



3. Skapa en egendefinerad funktion *antal_kameror(plats_data)* som räknar ihop antal kameror i varje kommun, sorterar resultatet i alfabetisk ordning efter kommun och presenterar resultatet som en tabell. Ni behöver inte ta hänsyn till om å, ä och ö sorteras fel, det kan bero på felaktig ordning i aktuell teckenkodning. I slutet ska även det totala antalet kameror summeras. Avsluta uppgiften med ett huvudprogram som visar att din funktion fungerar korrekt. Tabellen ska strukturellt se ut som nedan (obs värdena nedan är på denna deluppgift bara påhittade, så att du själv får träna på att kontrollera att dina resultat verkar stämma):

```
Kommun       Antal kameror
---------------------------
Alingsås             17
Bengtsfors            4
Essunga              10
Falköping            28
 …
Vara                  7
Vänersborg           20
Vårgårda             13
---------------------------
Det finns totalt 297 kameror.
```

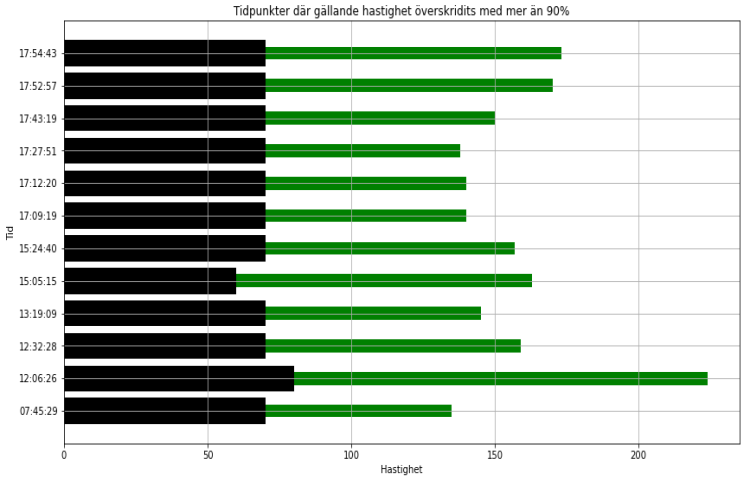
4. **a)** Skapa en egendefinerad funktion *hastighetplus(kamera_data)* som tar fram information om sådana fall där hastigheten var ett av användaren vald procentsats större än *Gällande Hastighet*. Funktionen börjar med att fråga efter procentsatsen och funktionen plockar sedan ut de poster där överträdelsen överstiger angiven procentsats. Lagra dessa poster i en lista kallad *plushastighet.* Listan skall sorteras efter *Tid* och nedanstående information och tabell skrivs ut. Funktionen avslutas med att returnera listan *plushastighet*, samt variabeln *procentover* som var användarens inmatade procentsats. Tabellen nedan visar resultatet för 90% men funktionen skall givetvis fungera för även andra procentsatser. Observera att *Tid*-kolumnen ska flyttas längst till vänster i denna tabellutskrift. *Kursiv text anger värden inmatade vid programkörning*

```
Söker ut alla poster där den procentuella hastighetsöverträdelsen
överskrider gällande hastighet med mer än vald %-sats.
Med hur många % skall hastigheten överskrida hastighetsbegränsningen för
att visas? 90

Det var 12 överträdelser som var mer än 90% över gällande hastighet.
```

Tid	MätplatsID	Gällande Hastighet	Hastighet	Datum
07:45:29	14039010	70	135	2021-09-11
12:06:26	14007040	80	224	2021-09-11
12:32:28	14090060	70	159	2021-09-11
13:19:09	14090050	70	145	2021-09-11
15:05:15	14048010	60	163	2021-09-11
15:24:40	14052030	70	157	2021-09-11
17:09:19	14093020	70	140	2021-09-11
17:12:20	14066040	70	140	2021-09-11
17:27:51	14093040	70	138	2021-09-11
17:43:19	14052010	70	150	2021-09-11
17:52:57	14052040	70	170	2021-09-11
17:54:43	14052010	70	173	2021-09-11

b) Skapa en egendefinerad funktion *hastighetplusdiagram(plus_hastighet, procent_over)* som ritar ett stapeldiagram likt det nedan utifrån en returnerad överträdelselista *plushastighet* och överträdelseprocensats *procentover* från a). Den svarta delen av stapeln representerar vad som var den tillåtna hastigheten, och grönt hur stor överträdelsen var.

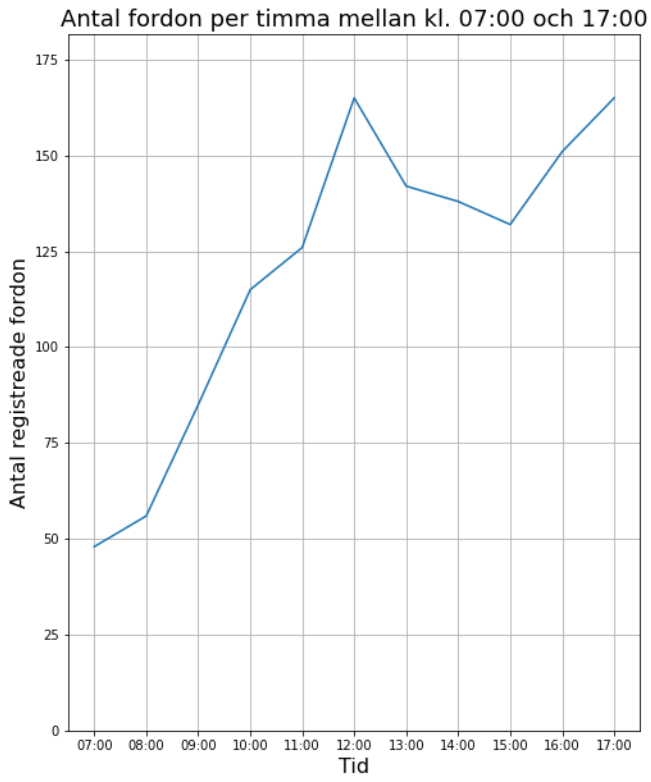


c) Skapa en egendefinerad funktion *hastighetkommun(plus_hastighet, plats_data)* som specificerar vägnummer och kommun för överträdelserna med hjälp av informationen i de två (underförstådda typer av) listor i inargumenten. Resultatet ska skrivas ut i en tabell likt nedan:

```
Hastighetsöverträdelserna skedde vid följande 12 platser.
Tid        MätplatsID       Kommun        Vägnummer     Hastighet
-------------------------------------------------------------------
07:45:29    14039010       Skara              49          135
12:06:26    14007040       Mariestad          E20         224
12:32:28    14090060       Strömstad         1040         159
13:19:09    14090050       Strömstad         1040         145
15:05:15    14048010       Tjörn              169         163
15:24:40    14052030       Göteborg           587         157
17:09:19    14093020       Göteborg         E6.21         140
17:12:20    14066040       Kungälv            625         140
17:27:51    14093040       Göteborg         E6.21         138
17:43:19    14052010       Göteborg           587         150
17:52:57    14052040       Göteborg           587         170
17:54:43    14052010       Göteborg           587         173
```

Avsluta uppgift 4 med ett huvudprogram som visar att dina funktioner från a), b), och c) fungerar korrekt.

5. Skapa en egendefinerad funktion *bilar_timme(kamera_data)* som ritar nedan diagram. Diagrammet visar summan av antal fordon som kamerorna har registrerat över alla mätplatser i alla kommuner uppdelat per timma mellan kl. 07 och 17. I diagrammet är x-axeln graderad i timmar mellan 07:00 och 17:00 och y-axeln i antal registrerade fordon, även rubrikerna skall vara med enligt nedan. Tips, använd den tillåtna modulen *datetime* i denna deluppgift.



Vi får lov att komma ihåg att detta diagram blir väldigt konceptuell i och med det nedbantade kameraData.csv-dataset:et. Verkligt antal bilar som åkte denna dag är snarare i storleksordningen ca 50000(!) per timma, jämfört med ca de 100 som vi får här

6. Skapa ett sammanhållet program som kan kalla på dina skrivna funktioner i deluppgift 1-5, likt nedan:

```
1. Hämta data från fil (obs måste göras först 1 gång). [Upg1]
2. Analysera data
      a. Antal bilar. [Upg2]
      b. Antal kameror. [Upg3]
      c. För fort. [Upg4]
      d. Bilar per timme. [Upg5]
3. Avsluta

Välj menyalternativ (1-3):
```

Efter att ett menyalternativ är valt (plus ev. undermeny) och detta är utfört, ska användaren kunna börja om med att ange ett nytt menyalternativ. Dvs det ska loop:as tills användaren väljer att avsluta. Menyval 2 innehåller en undermeny som skrivs ut först för att välja vad som skall analyseras.

Funktionsbeskrivning av menyalternativen:

1. Frågar användaren efter sökväg & namn till de två aktuella datafilerna. Skriver man inget utan bara trycker Enter här så används standardnamnen *kameraData.csv. och platsData.csv* (och förutsätter att de ligger i samma mapp som Jupyter-filen på datorn). Men det ska alltså även fungera att placera sina datafiler i t.ex. en undermapp förhållande till var din *”Svarsfil_inluppg_A237TG.ipynb”* ligger, och även döpta till något annat. *På kamratgranskningen så glöm inte att testa att detta funkar!*

   Tips: använd input-varianten i stil med detta:
   `input("sökväg & namn för kameraData.csv: ") or "kameraData.csv"`
   …för att lätt uppnå denna funktionalitet

   Sen kallas *read_file()* två gånger med dessa datafiler som argument, och sparar returnerade listorna i variabler döpta *kameradatameny* respektive *platsdatameny.* Skriv ut som i deluppgift 1 ut de 3 första raderna från filerna, för att bekräfta för användaren att filerna lästes in korrekt innan man går vidare till analyserandet i menyval 2

2. Användaren får skriva in igen vilket undermenyval som önskas (a-d). Sen kallas respektive funktion från deluppgift 2-5 beroende på val och dess resultat skrivs ut, med de inlästa listorna *kameradatameny* och *platsdatameny* från menyval 1 som argument. På deluppgift 4 så ska alla tabeller/diagram från 4a), 4b) 4c) skrivas ut.

3. Avslutar programmet (meny-evighetsloop:en bryts), *”Programmet avslutas”* eller liknande bekräftelse skrivs ut till användaren först.